DARPA/ISTO QUARTERLY REPORT

CONTRACT NUMBER:	N00014-88-K-0548
CONTRACTOR:	Duke University
ARPA ORDER NO:	4331714-01/4-6-88
PROGRAM NAME:	N00014
CONTRACT AMOUNT:	696,899.00
EFFECTIVE DATE OF CONTRACT:	July 1, 1988
EXPIRATION DATE OF CONTRACT:	June 30, 1991
PRINCIPAL INVESTIGATOR:	John H. Reif
TELEPHONE NUMBER:	(919) 660-6568
SHORT TITLE OF WORK:	Parallel Algorithms Derivations
REPORTING PERIOD:	April 1, 1989 - June 30, 1989

DARPA/ISTO Interim Technical Report[1]
Parallel Algorithm Derivation
DARPA order number: 4331714-01/4-6-88
Contract number: N00014-88-K-0458
Effective date: July 1, 1988
Expiration date: June 30, 1991

Principal Investigator: John Reif
Duke University
Durham, NC 27706
(919)-660-6568

July 1, 1989

[1]Notice of Disclaimer: The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

Chapter 1

REPORT SUMMARY

1.1 Purpose of project

We are developing a system of rules and techniques for derivation of various classes of parallel algorithms including:

1. Systolic algorithms for various fixed connection networks
2. Randomized parallel algorithms
3. Parallel algorithms for tree and graph problems
4. Parallel algorithms for algebraic problems

We are emphasizing the development of fundamental derivation techniques that can be utilized in as wide a class of parallel algorithms as possible. The specific algorithms to be derived have themselves been carefully chosen to be as fundamental as possible. Algorithms and areas currently under investigation include: parallel list ranking, parallel graph connectivity, automatic parallel compilation from segmented straight-line programs, and derivation of pipelined algorithms for small-diameter networks. A textbook, "Synthesis of Parallel Algorithms" (to be edited by Reif) is under way. A list of talented and well-known authors have agreed vist Duke in the Fall of 1989, and many are collaborating with Reif on their chapters. This text should bring together ideas from many sources on derivation of parallel algorithms. A new programming language, intended to be used as a Common Prototyping Language is under development.

We intend that the derivation techniques developed in this project will be implemented within the various other DARPA sponsored derivation systems.

1.2 Equipment Purchased

The following important items of equipment were purchased under this contract:

DESCRIPTION OF PROGRESS

Investigations of several subproblems in the area of derivation of parrallel programs were continued during the current quarter. These investigations include:

1. Derivation of various parallel algorithms, parallel graph connectivity and parallel list ranking (with student, Doreen Yen); parallel list ranking is now complete, now working on parallel graph connectivity;
2. Automatic Parallel compilation from segmented straight line programs (with student, Lorrie Tomek),
3. Derivation of pipelined algorithms on small networks (with student, Steve Tate and Prof. Robert Wagner),
4. Programming Language: Common Prototype Language (CPL), developed by student Lars Nyland and Professors Jan Prins, Robert Wagner and John Reif. CPL uses UNITY Primitives; now collaborating with Kestrel Institute with DARPA Proposal in CPL;
5. Duke algorithm Derivation Semninar: participants–Professors Robert Wagner, Donald Loveland, Gopalan Nadathur and John Reif; four visiting quest speakers in attendance were Professors Guy Blelloch, Gary Miller, Yijie Han and Victor Pan;
6. Proceeding on a textbook on "Synthesis of Parallel Algorithms" (to be edited by Reif) participating authors are to visit Duke University in Fall 1989, many of whom are collaborating with Reif on their chapters.

July 24, 1989

LIST OF EQUIPMENT PURCHASED

5 EACH	MACINTOSH COMPUTERS	18,258.87
5 EACH	80 MB INTERNAL MACUSE	4,547.00
4 EACH	RADIUS 19" MONITORS	8,040.00
1 EACH	0030 RADIUS 19" MONITOR	3,020.00
1 EACH	MATHEMATICA FOR MACKII	675.75
1 EACH	APPLE LASERWRITER PRINTER	3,350.00
	TOTAL EXPENSED TO DATE	37,891.62
	TOTAL AWARDED EQUIPMENT	(84,000.00)
	BALANCE IN EQUIPMENT NOT SPENT	(46,108.38)

1.3 Conclusions

Work is ongoing as described in the technical results section of this report.

Chapter 2

Technical Results

2.1 Textbook: Synthesis of Parallel Algorithms

Reif has invited a large number of prestigious researchers in the field of Parallel Algorithms to participate in writing a textbook on algorithms synthesis. The text is intended to review the existing knowledge in this area, at the level of a Senior undergraduate or first year graduate student. This text should draw together the many different principles which have been used to develop the current large collection of parallel algorithms which are theoretically interesting.

At the present writing, some 31 researchers have agreed to submit chapters of this text, and to referee 2 other chapters. Reif intends to collaborate with several of these researchers, and has invited them to visit Duke, where they will be available for discussion with other members of the Duke community, including the participants in the other projects funded by this contract.

This textbook promises to have significant impact on the development of parallel algorithms in the future. It should also serve as a central source, from which the details of the derivation process for some classes of algorithms can be extracted, and turned into a toolset useful for developing future algorithms.

In inviting participation, Reif suggested that each chapter begin with a careful statement of the fundamental problems, and the solution and analytic techniques to be used in their solution. He suggested that these techniques be related, where possible, to known sequential efficient algorithms. In later sections of the chapter, more sophisticated parallel algorithms are to be synthesized from the simpler parallel algorithms and techniques discussed earlier. Thus, a progression from simple to more complicated (and presumably more efficient) algortihms would be created. This progression should reveal the kinds of transformations needed in synthesizing this type of algorithm.

2.1.1 Participating Authors and Topics

1.	RICHARD LADNER University of Washington	PARALLEL PREFIX COMPUTATION
2.	GUY BLELLOCH Carnegie-Mellon University	MULTIPREFIX COMPUTATIONS AND APPLICATIONS

3. GARY MILLER
Carnegie-Mellon University
PARALLEL TREE CONTRACTION AND APPLICATION

4. PARALLEL CONNECTIVITY AND LIST RANKING

A. SARA BAASE
San Diego State University
INTRODUCTION TO PARALLEL CONNECTIVITY, LIST RANKING AND EULER TOUR TECHNIQUE

B. UZI VISHKIN
UMIACS
University of Maryland
ADVANCED PARALLEL CONNECTIVITY AND LIST RANKING

5. HILLEL GAZIT
Duke University
RANDOMIZED PARALLEL CONNECTIVITY

6. VIJAYA RAMACHANDRAN
University of Texas at Austin
PARALLEL EAR DECOMPOSITION WITH APPLICATIONS TO PARALLEL BICONNECTIVITY AND TRICONNECTIVITY

7. VIJAY VAZIRANI
Cornell University
PARALLEL GRAPH MATCHING

8. ERICH KALTOFEN
RPI
DYNAMIC PARALLEL EVALUATION OF DAGs

9. JEFFREY ULLMAN
Stanford University
PARALLEL EVALUATION OF LOGIC QUERIES

10. RAO KOSARAJU
The Johns Hopkins University
TOPIC: TO BE ASSIGNED

11. LARRY RUZZO
University of Washington
PARALLEL CFL AND DCFL RECOGNITION

12. PHILIP KLEIN
Harvard University
PARALLEL PQ TREE OPERATIONS AND APPLICATIONS

13. MICHAEL LUBY
International Computer Science Institute
PARALLEL COMPUTATION OF MAXIMAL INDEPENDENT SET

14. VICTOR PAN CUNY	PARALLEL SOLUTION OF SPARSE LINEAR AND PATH SYSTEMS
15. ANDREW GOLDBERG Stanford University	PARALLEL NETWORK FLOW
16. PAUL BEAME University of Washington	PARALLEL CHINESE REMAINDERING AND INTEGER DIVISION
17. STEPHEN TATE Duke University	NEWTON ITERATION AND INTEGER DIVISION
18. MICHAEL BEN-OR Hebrew University	PARALLEL ROOT FINDING
19. JOACHIM von zur GATHEN University of Toronto	PARALLEL MATRIX COMPUTATIONS AND SOLUTION OF ALGEBRAIC PROBLEMS
20. NIMROD MEGIDDO IBM Almaden Res. Ctr.	PARALLEL LINEAR PROGRAMMING
21. RICHARD COLE Courant Institute	PARALLEL MERGE SORT
22. MIKHAIL ATALLAH Purdue University	DETERMINISTIC PARALLEL COMPUTATIONAL GEOMETRY
23. SANGUTHEVAR RAJASEKARAN University of Pennsylvania	RANDOMIZED PARALLEL SELECTION AND SORTING
24. SANDEEP SEN Duke University	RANDOMIZED PARALLEL COMPUTATIONAL GEOMETRY
25. RICHARD ANDERSON University of Washington	PARALLEL DEPTH FIRST SEARCH
26. ZVI GALIL Columbia University	PARALLEL STRING MATCHING

27. RICHARD KARP University of California	PARALLEL TREE SEARCH
28. ALOK AGGARWAL and JAMES PARK IBM Watson Res. Ctr.	PARALLEL DYNAMIC PROGRAMMING
29. PHIL GIBBONS University of California	ASYNCHRONOUS PRAM ALGORITHMS
30. RAYMOND GREENLAW University of Washington	POLYNOMIAL COMPLETENESS AND PARALLEL COMPUTATION
31. BARUCH SCHIEBER IBM Watson Res. Ctr.	PARALLEL LOWEST COMMON ANCESTOR COMPUTATIONS

2.2 CPL: A High-level Prototyping Language

We are developing a language to be used for prototyping. The goal is to facilitate the initial prototyping of algorithms as executable programs. The Common Prototype Language (CPL) is our proposed answer to this goal. Our view of prototyping is the ability to write programs primarily for the purpose of experimentation and validation of ideas in a quick and easy fashion. Prototype programs do not necessarily have complete specifications.

2.2.1 The Essence of CPL

DARPA/ISTO has challenged the community to design a prototyping language with ambitious capabilities. We propose a Common Prototyping Language (CPL) that meets this challenge. The essence of our proposal is to:

- Adopt UNITY[1], a programming paradigm for developing algorithms on parallel and distributed computers as the fundamental model of control in CPL. UNITY is supported by a programming logic, which allows formal reasoning about the correctness, safety and progress of computations. The UNITY style supports formal program development from abstract specifications to concrete implementations.
- Provide CPL with a rich data model incorporating set-theoretic type constructors such as sets and relations, objects, user-defined data abstractions, and a rich, flexible notion of type.
- Augment the control structures of UNITY to support block structure, subroutines, sequencing, iteration, AI programming methods, and modules. These extensions will be consistent with the underlying semantics and will provide a surface syntax similar to conventional languages.

- Adopt REFINE's [2] language extension capabilities, which supports the definition of grammars, parsing and printing, a standard representation of abstract syntax within the data model, and a powerful pattern and transformation capability.

CPL is derived from UNITY[1], a programming paradigm for developing algorithms on parallel computers. Although the CPL has only a small number of control primitives, they have the expressive power to describe sequencing, concurrency, parallel computation, and nondeterministic search by forward and backward chaining. CPL will support a small number of built in fundamental classes such as sets, arrays, and relations, with correspondingly rich set of standard mathematical operators and syntax. CPL allows various levels of specification, from very high level to much lower level, with environment support for derivation of programs at various levels of specification. For example, we propose a flexible type system with type inference that allows transition from typeless declarations to fully typed programs.

CPL demands high-level primitives over complex data types to support the mathematical notions of execution, and descent to low-level primitives such as variables, assignments and machine-oriented data types (integers vs. reals) to model the more concrete aspects. We believe that the best way to proceed with CPL is to incorporate as much of REFINE[2], a high-level program development model based on program transformation, as is consistent with the requirements for a prototype language, and to rely on it to provide a wide-spectrum of constructs to control execution and state.

2.2.2 Our Team

Our team has broad experience within the fields of languages, systems, distributed and parallel computation, compilers and derivational computing. Experience with REFINE, a system which approaches many of the design goals of CPL, will prove invaluable.

The group of people involved with this project come from three institutions: Duke University, The University of North Carolina, and Kestrel Institute. Professors John Reif (Ph.D. from Harvard on Symbolic Program Execution and an expert in parallel computation and algorithm derivation) and Robert Wagner (Ph.D. from Carnegie-Mellon University on Compiler Optimization and an expert on parallel computation and compilers) are defining the project with help from graduate student Lars Nyland. They are working in cooperation with UNC professor Jan Prins (Ph.D. from Cornell University on Programming Languages and an expert in derivational computing). From Kestrel, Allen Goldberg (Ph.D. from Courant Institute, and an expert on program derivation and optimization), Richard Jüllig (Ph.D. from University of California, Santa Cruz, an expert on compilers and meta-compilers), and Doug Smith (Ph.D. from Duke, an expert on algorithm design and program transformation) will be working on the project.

Bibliography

[1] K. Mani Chandy and Jayadev Misra. *Parallel Program Design, A Foundation.* Addison-Wesley Publishing Company, 1988.

[2] Reasoning Systems, Inc., 1801 Page Mill Rd., Palo Alto, CA 94304, (415) 494-6201. *Refine User's Guide*, August 1988. For Refine version 2.0 with updates for version 2.1 on Sun-3 series computer.

2.3 Duke Algorithm Derivation Seminar

During Fall, 1988, a seminar on algorithm derivation was held at Duke. Participants included Don Loveland, Hillel Gazit, Gopalan Nadathur, Robert Wagner, and John Reif, all professors at Duke; Four visiting guest speakers were Professors Guy Blelloch, Gary Miller, Yijie Han, and Victor Pan. Topics covered included papers by Marina Chen and D. Moldovan concerning derivation of Systolic Arrays from recurrence equations, papers by Chen on the CRYSTAL language for expressing algorithms as recurrence equations, papers by Reif on deriving randomized algorithms, papers by Scherlis, Burstall and Darlington, and Paige on derivation of algorithms by program transformations. The Duke participants prepared lectures on these topics, which were presented in one or two meetings each, in a setting which allowed for considerable discussion.

Paige's work on derivation of algorithms by algorithm finite differencing was presented by his student, Jiazhen Cai. Each of the other visitors presented a formal lecture, whose titles are given in the subsection which follows. In addition, some of the visitors discussed some of their earlier work with the Duke participants. Blelloch talked on the use of Parallel Prefix (Scan) operations in parallel algorithms, Miller discussed his work on efficient parallel evaluation for DAGs.

Yijie Han presented a lecture on deriving an efficient algorithm for the linked-list parallel prefix problem on a network of processors with restricted communication access, and Victor Pan presented some work on Computing polynomial zeros and its impact on matrix eigenvalue computation. . This seminar served to introduce the Duke participants to the areas with which they were unfamiliar, and served to suggest directions for further work for the group. A principal outcome of the seminar was a realization that progress in algorithm derivation depends strongly on the language in which algorithms expressed initially, and lead to some discussions of a possible universal language for expressing parallel algorithms in a succinct, understandable form, independent of any particular parallel architecture. Discussions on this topic were joined by Professor Jan Prins, of UNC, and have resulted in considerable work in defining such a language, which we deem usable as a "Common Prototyping Language".

2.3.1 Lectures by Visitors

February 14, 1989	Guy E. Blelloch, Carnegie Mellon University Compiling Collection-Oriented Languages Onto Massively Parallel Computers
March 22, 1989	Victor Pan, SUNY, Albany New Progress in Computing Polynomial Zeros And Its Impact on Matrix Eigenvalue Computation
March 23, 1989	Victor Pan, SUNY, Albany Parallel Inversion and Factorization of Toeplitz and Toeplitz-Like Matrices
March 31, 1989	Gary Miller, Carnegie Mellon University Optical Communication For Pointer Based Algorithms
May 5, 1989	Yijie Han, University of Kentucky An Optimal Linked List Prefix Algorithm On A Local Memory Computer

2.4 Synthesizing Algorithms for Small Diameter Networks

Small diameter networks (such as the hypercube and the butterfly) are becoming increasingly popular as an interconnection network for large multiprocessors. While much work has been done developing parallel algorithms that run on such networks, the vast majority of work has focused on particular algorithms. For simpler networks (such as square and hexagonal grids), there are many papers on general derivation techniques. We propose similar derivation results for the small diameter networks; the methods seem to be efficient enough for effective implementation.

2.4.1 Small Diameter Networks

A network is simply an interconnection of processing units. One of the most important measures of a network is its diameter; the diameter is simply the longest distance between two processors in the network (measured as the number of intermediate processors that the path must go through). Obviously, the most efficient network is one in which every processor is directly connected to ever other processor (for a diameter of 1); unfortunately, this connection strategy becomes too complex to be feasible for even moderately large multiprocessors.

On the other hand, the most easily visualized (and implemented) network is a square grid (every processor has exactly 4 neighbors, regardless of the total number of processors). As can be easily seen, for a network of n processors, the diameter is $\sqrt{n}$.

Other types of networks have been proposed that have diameter $O(\log n)$ for n processor networks, this diameter being between the unrealistic full connection network and the inefficient grid. Furthermore, the feasibility of these networks is shown by a number of hardware implementations (the BBN Butterfly and the Connection Machine are two examples). As an example of the improvement in diameter, a network with 64k (approximately 64,000) processors connected by a square grid has diameter 256 while the same number of processors connected by a hypercube has a diameter of only 16. This means that certain algorithms on a hypercube of 64k processors will run 16 times faster than the algorithm would run on a grid with the same number of processors. The difference becomes even more extreme as larger networks are considered (with approximately a million processors, the grid diameter is 50 times greater than the diameter of the hypercube).

2.4.2 Derivation of Algorithms

There have been many papers on mapping algorithms to the simple architectures of planar grids (square or hexagonal); see, for example, [2], [3], [7], and [8]. Unfortunately, small diameter networks are more difficult to describe in uniform mathematical terms using previous methods; one attempt to find a unifying framework for small diameter networks can be found in [1].

We note that most (in fact, *all* that we know of) small diameter networks have a natural representation using the abstract algebra concept of a ring of polynomials over a field (most common networks are described using $Z_2[X]/(X^{\log n}+1)$). From abstract algebra, we also know that many of the operations used in the previous algorithms for mapping to grids can now be used in our new domain, giving us a way of mapping algorithms to small diameter networks.

We do not claim to map all algorithms to small diameter networks; however, what we do claim is that we have the first *automatic* method of mapping algorithms to small diameter networks. It is our hope that this will give rise to additional research that broadens its usefulness (as was the case with research on mapping algorithms to grid networks).

2.4.3 Implementation

Our mapping methods should be practical enough to be implemented — we plan on implementing our results in some portable programming language, and then testing how practical these methods actually are. The implementation will be similar to existing software that maps algorithms to grids (e.g., ADVIS [4]), but doing calculations over our new algebraic domain.

2.4.4 Researchers

This project is being run by John Reif, Ph.D., at Duke University. Also involved in the entire project is Robert Wagner, Ph.D., who is also at Duke. The theoretical background is being worked on by graduate student Stephen Tate (Duke University), and implementation will be done by Akitoshi Yoshida, also a graduate student at Duke University.

Bibliography

[1] F. Annexstein, M. Baumslag, A.L. Rosenberg. "Group Action Graphs and Parallel Architectures," *COINS Technical Report 87-133*, 1987.

[2] M.C. Chen. "A Design Methodology for Synthesizing Parallel Algorithms and Architectures," *Journal of Parallel and Distributed Computing*, 1986, pp. 461-491.

[3] D.I. Moldovan. "On the Design of Algorithms for VLSI Systolic Arrays," *Proceedings of the IEEE*, Vol. 71, No. 1, (Jan 1983), pp. 113-120.

[4] D.I. Moldovan. "ADVIS: A Software Package for the Design of Systolic Arrays," *ICCD*, 1984, pp. 158-164.

[5] D.I. Moldovan, and J.A.B. Fortes. "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," *IEEE Transactions on Computers*, Vol. 35, No. 1, (Jan 1986), pp. 1-12.

[6] F.P. Preparata, and J. Vuillemin. "The Cube Connected Cycles: A Versatile Network for Parallel Computation," *CACM*, Vol. 24, No. 5, (May 1981), pp. 300-309.

[7] P. Quinton. "Automatic Synthesis of Systolic Arrays ¿From Uniform Recurrent Equations," *Proc. 11th International Symposium on Computer Architecture*, 1984, pp. 208-214.

[8] S.V. Rajopadhye, S. Purushothaman, R.M. Fujimoto. "On Synthesizing Systolic Arrays from Recurrence Equations with Linear Dependencies," *Proc. of Sixth Conference on Foundations of Software Technology and Theoretical Computer Science*, (Dec 1986), pp. 488-503.

2.5 Derivation of parallel algorithms

There are a sufficient number of parallel random-access machine (PRAM) algorithms to consider parallel algorithm program derivation. A primary reference for these algorithms is the survey by Karp. [3] Several problems such as list ranking have spawned a series of increasingly efficient algorithms, both randomized [6] and deterministic [1] which provide examples for an application of the evolutionary approach to program transformations advocated by Reif and Scherlis. [7] Some PRAM algorithms are closely related. Expression evaluation on trees is reducible to list ranking. [2] The problem of computing the connected components of a graph has a similar history. As in the sequential graph algorithm derivations by Reif and Scherlis, the PRAM algorithm domain offers the opportunity for reasoning by analogy to derive a family of related algorithms.

UNITY [5] will be considered as a programming language for PRAM algorithm derivation. Experiments on PRAM algorithms with program transformation steps used in the ERGO project: [4] reduction, expansion, choice-function introduction, and choice-function elimination will be conducted but other transformation steps peculiar to PRAM algorithms will be needed. For example, a common technique in the evolution to an optimal PRAM algorithm requires finding a way to divide the input into size log n chunks.

Bibliography

[1] Richard J. Anderson and Gary L. Miller. Deterministic parallel list ranking. 1988.

[2] H. Gazit, Gary L. Miller, and S.H. Teng. Optimal tree contraction in the EREW model.

[3] Richard M. Karp and Vijaya Ramachandran. *Handbook of Theoretical Computer Science*, chapter A Survey of Parallel Algorithms for Shared-Memory Machines. North-Holland.

[4] Peter Lee, Frank Pfenning, John Reynolds, Gene Rollins, and Dana Scott. Research on semantically based program-design environments: The Ergo Project in 1988. Technical Report CMU-CS-88-118, Carnegie Mellon University, March 1988.

[5] Chandy K. Mani and Jayadev Misra. *Parallel Program Design: A Foundation.* Addison-Wesley Pub. Co., 1988.

[6] Gary L. Miller and J.H. Reif. Parallel tree contraction and its applications. *26th Symposium on Foundations of Computer Science*, pages 230 – 239, 1985. Randomized algorithm for list ranking.

[7] J.H. Reif and W.L. Scherlis. Deriving efficient graph algorithms. Sequential algorithm derivations, 1982.

2.6 Randomized Algorithms

Reif and his student, Sandeep Sen, have developed several randomized algorithms, and have studied the process by which such algorithms are developed. Several papers [1, 2, 3, 4] have been produced. There are clearly some general principles underlying the derivation of such algorithms, and these issues will be explored in followup work.

Bibliography

[1] J. Reif, S. Sen. "Optimal Parallel Algorithms in Computational Geometry" *1987 International Conference on Parallel Processing*, University Park, PA, Aug. 1987.

[2] J. Reif, S. Sen. "An efficient output-sensitive hidden-surface removal algorithm and its parallelization" *4th Annual Computational Geometry Conference*, 198-200, Urbana-Champaign, IL, June 1988.

[3] J. Reif, S. Sen. "Polling: A New Randomized Sampling Technique for Computational Geometry" *21st Annual ACM Symposium on Theory of Computing*, Seattle, WA, May 1989.

[4] J. Reif, S. Sen. "A Case for Randomized Parallel Algorithms" *Workshop on Capabilities and Limitations of Parallel Computing*, IBM, San Jose, CA, Dec. 1988.